Implicit Structural Analysis of Multimode DAE Systems

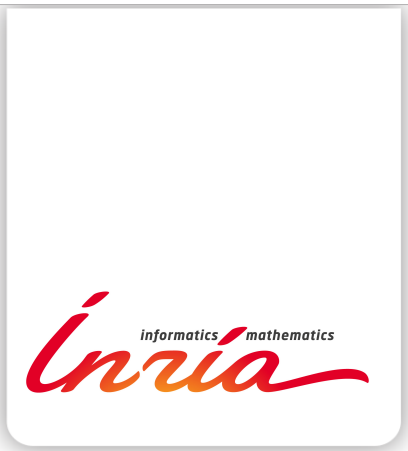
Benoît Caillaud, Mathias Malandain, Joan Thibault

Implicit Structural Analysis of Multimode DAE Systems

Benoît Caillaud, Mathias Malandain, Joan Thibault

RESEARCH REPORT
N° 9322
February 2020
Project-Team Hycomes

ISSN 0249-6399 ISRN INRIA/RR--9322--FR+ENG



Implicit Structural Analysis of Multimode DAE Systems

Benoît Caillaud, Mathias Malandain, Joan Thibault

Project-Team Hycomes

Research Report n° 9322 — February 2020 — 35 pages

Abstract: The Modelica mathematical modeling language, based on Differential Algebraic Equations (DAE), brings several specific issues that do not exist with modeling languages based on Ordinary Differential Equations. The main problem is the determination of the differentiation index and latent equations. Prior to generating simulation code and calling solvers, the compilation of a Modelica model requires a structural analysis step, which reduces the differentiation index to a level acceptable by numerical solvers.
The Modelica language allows hybrid models with multiple modes, mode-dependent dynamics and state-dependent mode switching. These Multimode DAE (mDAE) systems are much harder to deal with. The main difficulties are (i) the combinatorial explosion of the number of modes, and (ii) the correct handling of mode switchings.
The focus of this report is on the first issue, namely: How can one perform a structural analysis of an mDAE in all possible modes, without enumerating these modes? A structural analysis algorithm for mDAE systems is presented, based on an implicit representation of the varying structure of an mDAE. It generalizes J. Pryce's structural analysis method to the multimode case and uses Binary Decision Diagrams (BDD) to represent the mode-dependent structure of an mDAE. The algorithm determines, as a function of the mode, the set of latent equations, the leading variables and the state vector. This is then used to compute a mode-dependent block-triangular decomposition of the system, that can be used to generate simulation code with a mode-dependent scheduling of the blocks of equations.
This report is an extended version of the homonym paper, published in the proceedings of the HSCC'20 conference [7].

Key-words: structural analysis, differential-algebraic equations (DAE), multi-mode systems, variable-structure models, binary decision diagrams (BDD)

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Analyse structurelle implicite des systèmes de DAE multimodes

Résumé : Le langage de modélisation mathématique Modelica, basé sur les systèmes d'équations algébro-différentielles (Differential Algebraic Equations, ou DAE), présente des difficultés qui n'apparaissent pas dans la modélisation à base d'équations différentielles ordinaires. Le problème principal consiste à déterminer l'indice de différentiation et les équations latentes d'un système. Lors de la compilation d'un modèle Modelica, un prérequis à la génération de code de simulation et à l'appel de solveurs est une étape d'analyse structurelle, qui réduit l'indice de différentiation à un niveau acceptable par les solveurs numériques.

Le langage Modelica permet l'écriture de modèles hybrides, ou multimodes, dont la dynamique dépend du mode, et où les changements de mode sont conditionnés par les variables d'état. Ces DAE multimodes (mDAE) sont considérablement plus difficiles à traiter, les obstacles principaux étant (i) l'explosion combinatoire du nombre de modes, et (ii) la prise en charge des changements de mode.

Ce rapport se focalise sur le premier point: Comment effectuer l'analyse structurelle d'une mDAE en prenant en compte tous ses modes, mais sans énumérer ces derniers ? Un algorithme d'analyse structurelle de systèmes de mDAE est proposé; il s'appuie sur une représentation implicite de la structure variable de la mDAE considérée. La méthode présentée généralise celle de Pryce au cas multimode, et utilise des Diagrammes de Décision Binaire (Binary Decision Diagrams, ou BDD) pour représenter la structure variable d'une mDAE. L'algorithme détermine l'ensemble des équations latentes, les variables de tête et le vecteur d'état, en tant que fonctions du mode. Ces informations permettent ensuite de décomposer le système en une forme triangulaire par blocs dépendant du mode. Cette dernière peut, par la suite, être utilisée pour générer du code de simulation efficace, en prenant en compte un ordonnancement des blocs d'équations qui dépend du mode.

Ce rapport est une version étendue de l'article homonyme, publié dans les actes de la conférence HSCC'20 [7].

Mots-clés : analyse structurelle, équations algébro-différentielles (DAE), systèmes multi-mode, modèles à structure variable, diagrammes de décision binaires (BDD)

Contents

1 Introduction

Differential Algebraic Equations (DAE) are quite commonly used in multiphysics systems modeling. The main reason is that they enable a component-based modeling discipline, meaning that the way mathematical equations are organized precisely reflects the architecture of the physical apparatus or system they model. Large systems can be modeled with DAE, whereas this is a far more complex (if not daunting) task with *Ordinary Differential Equations* (ODE). This is easily conceivable by looking at the modular structure of physical models, that reflects the decomposition of a complex apparatus or system into possibly many elementary physical devices. The dynamics of each device is captured by a law, in the form of one or a few simple mathematical equations (algebraic relations and/or differential equations), and the interconnection of these devices results in the coupling of these laws by algebraic equations. While the structure of the resulting DAE is well understood for each particular physical domain (mechanics, electricity, thermics, etc.), multiphysics models come with an additional difficulty: the equations are not organized in a predefined way anymore, and the apparent structure of the DAE can be arbitrarily complex.

The Modelica[1] language is a mathematical modeling language based on DAE. Its object-oriented features make system modeling easier by mixing equation-based modeling with a component-based approach, where physical library components are instantiated and interconnected [14]. A difficulty with Modelica, and DAE in general, is that checking whether a model makes sense, from a mathematical point of view, is not an obvious question. Unlike ODE, it is indeed difficult to manually check whether a DAE is determined or nonsingular, i.e., that it admits a unique solution. It would be desirable to check this automatically; unfortunately, such a numerical property involves the actual values of model parameters, as well as, when considering nonlinear systems, the state of the model. Hence, this can only be done during simulation. However, tools supporting the Modelica language statically check, at compilation time, a necessary condition: the structural nonsingularity of a DAE. This concept, defined in the sequel of the paper, is supported both by solid mathematical foundations related to the concepts of latent equations and differentiation index of a DAE [8], and by well understood methods based, among others, on the renowned Pantelides algorithm [18] or the less known Σ-method [20]. This (static) structural analysis of a model is actually a necessary step for the generation of simulation code: using its output, Modelica tools[2] fill two needs with one deed, by providing users with a detailed diagnosis about the structure of the model at compile time, and by generating simulation code exploiting the sparsity of the model. In essence, the structural analysis of a DAE can be understood as analogous to the typechecking of a programming language: it flags obvious programming errors, possible misuse of the language (e.g., possible sources of bugs because of unsafe type operations), and is also required for code-generation.

Quite often, physical system models are not smooth. This can be a deliberate choice, by approximating a steep nonlinearity into a discontinuous behavior (e.g., contacts and Coulomb friction in multibody mechanics, switching of diodes and transistors), or a necessity, for instance when considering sudden faults (e.g., breakage of a mechanical linkage, shorting of an electrical component), dynamical system reconfigurations (connection/disconnection of electric vehicles to a power grid) or, in the case of Cyber-Physical Systems (CPS), because the physical system lives in close interaction with a computerized control, modeled as a time-/event-triggered discrete event system. Modeling such systems requires a more general mathematical framework. Allowing equations in a DAE to be activated/deactivated, depending on the state of a discrete control

[1] `https://www.modelica.org`

[2] Such as Dymola (`https://www.3ds.com/products-services/catia/products/dymola/`) and OpenModelica (`https://openmodelica.org`).

is a natural idea. This leads to the concept of *switched* or *multimode* DAE (mDAE) and the Modelica language has been extended in recent years to support such systems [13]. However, the mathematical foundations of mDAE are quite slim, and only cover either specific classes of mDAE, or particular control-theoretical problems on mDAE [17]. The structural analysis of mDAE is still in its infancy, with a few notable works on the characterization of their impulsive behavior [9, 3, 4].

From an algorithmic complexity point of view, analyzing the structure of an mDAE is very challenging as, usually, the number of modes is roughly exponential in the number of equations. Any method based on an enumeration of these modes would essentially be restricted to very small systems, and cancel out the scalability gained from the use of DAE. Of course, an alternative consists in performing the structural analysis at runtime, at every mode switching. This is the approach advocated in [5] and [12]. However, this approach does not allow to compile the model into an efficient code, and puts a heavy burden on the runtime: several computationally expensive steps have to be performed during simulation, including the index reduction of the active DAE, the automatic differentiation of some equations to produce the corresponding latent equations, then a block-triangular decomposition of the resulting system of equations.

This report proposes a novel, radically different approach, based on an implicit representation of the varying structure of an mDAE in all its modes. The first step of the method consists in determining the index and latent equations of the mDAE, as functions of the mode. The algorithm used here is a generalization of J. Pryce's Σ-method [20] to the multimode case. The second step is an adaptation of the Dulmage-Mendelsohn decomposition [19, 10] to compute the set of blocks of equations that can appear in some mode of the system. This is followed by a third and last step resulting in the computation of a conditional dependency graph, defining the dependencies between equation blocks in every possible mode. All these steps are performed all-modes-at-once on implicit functional representations; Binary Decision Diagrams (BDD) are used at every step of the method.

The end result is a structural analysis method for mDAE systems, that can be performed statically, at compile time.[3] If provided with a structurally unsound model, it returns precise diagnostics, pointing to possible errors; this gives the user the opportunity to correct his/her model before it is simulated. Otherwise, the conditional dependency graph computed by the method can be directly used to generate efficient simulation code, since it defines which equation blocks must be solved in each mode, and in what order.

The method has been implemented in OCaml in the IsamDAE software[4] and benchmarked on several examples of varying complexity.

Note that the method presented in this report only covers the structural analysis of the modes of an mDAE. Another facet of a complete multimode structural analysis would be the structural analysis of mode switchings, as considered in [3]. This is left for future work.

The paper is organized as follows. Multimode DAE (mDAE) systems are presented in Section 2, in informal terms. This section also introduces a simple running model, used in this report as an illustration of the concepts and algorithms introduced therein, and highlights limitations of the existing Modelica tools on multimode models. Section 3 recalls important elements of the DAE index theory, then presents the Σ-method and the block-triangular decomposition in the single-mode case. The original contribution is given in Section 4, namely: the definition of the considered class of mDAE systems, its encoding with Boolean functions, and the generalization of the Σ-method, followed by the mode-dependent block-triangular decomposition and the

[3] It extends to mDAE systems the algorithms currently implemented in Modelica tools, that only support single-mode DAE systems in a predictable and sound manner.

[4] IsamDAE can be tested online using the following link: `https://allgo18.inria.fr/apps/isamdae`

computation of the conditional dependency graph. Section 5 highlights experimental results on scalable models.

2 Multimode modeling

In this section, we informally introduce some of the basics of multimode DAE modeling thanks to a simple fixed-size model of an electronic circuit with diodes; this example is used in Sections 3 and 4 for illustrating the various algorithms and methods presented therein. We also show how this model is not properly handled by two leading Modelica tools, namely, Dymola and OpenModelica, and hint at reasons for this fact, justifying the approach developed in this report.

2.1 The RLDC2 model

A simple example of multimode DAE, courtesy of S. E. Mattsson, is the electronic circuit shown in Figure 1. This circuit is used in the sequel as an illustration of the different steps of structural analysis.

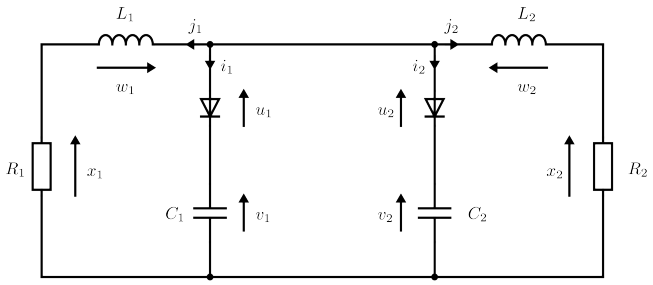


Figure 1: Schematics of the RLDC2 circuit.

The circuit consists in two RLC circuits interconnected in parallel and in which two diodes have been inserted. The diodes are considered to be ideal, meaning that they are not ruled by the customary Shockley law $i = I\left(e^{u/U} - 1\right)$, but rather by a complementarity condition $0 \leq i \perp -u \geq 0$, meaning that, at any moment, both i and $-u$ are nonnegative and $iu = 0$ (see Figure 2).

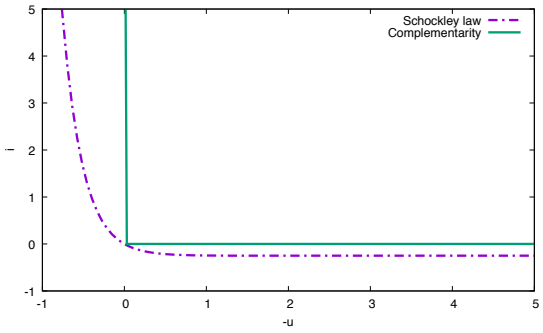


Figure 2: Shockley law vs. ideal complementarity condition.

Rather than using the formulation $-u, i \geq 0$ and $ui = 0$, the graph of a complementarity condition can be defined as a parametric curve, formalized as the following three equations:

$$\begin{array}{rcll} s & = & \text{if } g \text{ then } i \text{ else } -u & (S) \\ 0 & = & \text{if } g \text{ then } u \text{ else } i & (Z) \\ g & = & (s \geq 0) & (G) \end{array}$$

However, solving this system of equations requires computing fixed points of logico-numerical equations. This falls out of the scope of the paper, and instead of the above system, we consider the system obtained by replacing s with its left-limit s^- in equation (G):

$$\begin{array}{rcll} s & = & \text{if } g \text{ then } i \text{ else } -u & (S) \\ 0 & = & \text{if } g \text{ then } u \text{ else } i & (Z) \\ g & = & (s^- \geq 0) & (G^-) \end{array}$$

Under the assumption that u and i are continuous functions of time (which turns out to be a valid assumption for the RLDC2 circuit), $s(t) = s^-(t)$ holds at every instant t. This implies that systems $(S), (Z), (G)$ and $(S), (Z), (G^-)$ are equivalent. As explained in [2], the left-limit operator, when properly used, has the property of breaking algebraic loops. Indeed, equation (G^-) can be evaluated first and then, knowing the value of g, equations (S) and (Z) can be solved.

Using this encoding of the complementarity condition, the RLDC2 circuit is modeled as the mDAE shown in Figure 3, with two Boolean variables g_1 and g_2 and four multimode equations (S_1), (Z_1), (S_2) and (Z_2), incident to a varying set of variables depending on the mode of the system (i.e., the values of Boolean variables g_1 and g_2).

$$\begin{array}{rcll} 0 & = & i_1 + i_2 + j_1 + j_2 & (K_1) \\ x_1 + w_1 & = & u_1 + v_1 & (K_2) \\ u_1 + v_1 & = & u_2 + v_2 & (K_3) \\ u_2 + v_2 & = & x_2 + w_2 & (K_4) \\ w_1 & = & L_1 \cdot j_1' & (L_1) \\ w_2 & = & L_2 \cdot j_2' & (L_2) \\ i_1 & = & C_1 \cdot v_1' & (C_1) \\ i_2 & = & C_2 \cdot v_2' & (C_2) \\ x_1 & = & R_1 \cdot j_1 & (R_1) \\ x_2 & = & R_2 \cdot j_2 & (R_2) \\ s_1 & = & \text{if } g_1 \text{ then } i_1 \text{ else } -u_1 & (S_1) \\ s_2 & = & \text{if } g_2 \text{ then } i_2 \text{ else } -u_2 & (S_2) \\ 0 & = & \text{if } g_1 \text{ then } u_1 \text{ else } i_1 & (Z_1) \\ 0 & = & \text{if } g_2 \text{ then } u_2 \text{ else } i_2 & (Z_2) \\ g_1 & = & (s_1^- \geq 0) & (G_1^-) \\ g_2 & = & (s_2^- \geq 0) & (G_2^-) \end{array} \tag{1}$$

Figure 3: The RLDC2 model (*n.b.*: variable y' denotes the time derivative of y).

2.2 The RLDC2 with Modelica tools

The Modelica model of the RLDC2 circuit is given in Figure 4; it is a direct translation of the model above. The authors had the opportunity to test it on two implementations of the Modelica language: OpenModelica v1.12[5] and Dymola 2019.[6]

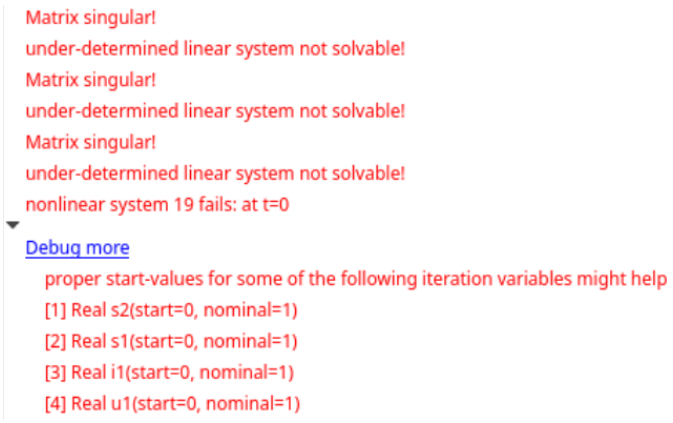
```
model RLDC2_CC "RLDC2 example with complementarity conditions"
  parameter Real R1 = 10.0;
  parameter Real R2 = 15.0;
  parameter Real L1 = 1.0;
  parameter Real L2 = 1.5;
  parameter Real C1 = 0.10;
  parameter Real C2 = 0.15;
  Real i1;
  Real i2;
  Real j1(start=2.0,fixed=true);
  Real j2(start=1.0,fixed=true);
  Real u1;
  Real u2;
  Real v1(start=0.5,fixed=true);
  Real v2(start=1.0,fixed=true);
  Real w1;
  Real w2;
  Real x1;
  Real x2;
  Real s1;
  Real s2;
  Boolean g1(start=false);
  Boolean g2(start=false);
equation
  0 = j1+i1+i2+j2; // (K1)
  x1+w1 = u1+v1; // (K2)
  u1+v1 = u2+v2; // (K3)
  u2+v2 = x2+w2; // (K4)
  x1 = R1*j1; // (R1)
  x2 = R2*j2; // (R2)
  w1 = L1*der(j1); // (L1)
  w2 = L2*der(j2); // (L2)
  i1 = C1*der(v1); // (C1)
  i2 = C2*der(v2); // (C2)
  s1 = if g1 then i1 else -u1; // (S1)
  s2 = if g2 then i2 else -u2; // (S2)
  0 = if g1 then u1 else i1; // (Z1)
  0 = if g2 then u2 else i2; // (Z2)
  g1 = (s1 >= 0); // (G1)
  g2 = (s2 >= 0); // (G2)
    annotation (...);
end RLDC2_CC;
```

Figure 4: Modelica model of the RLDC2 circuit.

[5] https://openmodelica.org
[6] https://www.3ds.com/products-services/catia/products/dymola/

Identical results are obtained with both tools: the model is deemed nonsingular at compile time, yet a runtime error immediately occurs (at time $t = 0$) during simulation, as shown in Figure 5.

(a)



(b)

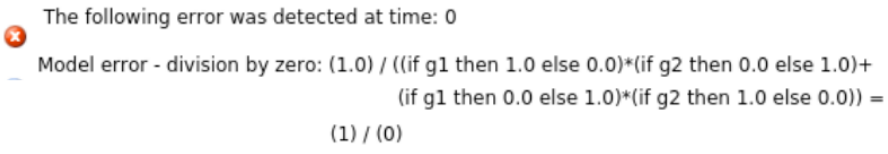


Figure 5: Error messages produced during simulation of the RLDC2 model with (a) OpenModelica and (b) Dymola, both at the initial time of the simulation.

The issue is that the structural analysis implemented in these tools treats the model as a single-mode DAE, disregarding the mode-dependent variability of the incidence relations. The consequence is that, despite the model being deemed structurally nonsingular by these tools, blocks of equations that are structurally singular in some modes are produced. This immediately leads to runtime errors: both tools attempt a pivoting of the Jacobian matrix by an element that is equal to zero.

This simple example proves the necessity of designing structural analysis tools that can properly handle multimode models, by accounting for the actual mode dependencies. As such an approach would both yield reliable diagnosis of structural singularity of an mDAE model in some or all of its modes, and enable the generation of efficient simulation code for every mode, it is akin to paving the way towards a compiler for mDAE models.

3 Preliminaries

This section focuses on the structural analysis and block dependency study of single-mode DAE. First, formal definitions of DAE systems are introduced; structural analysis is then presented, as a means of deriving an ODE system from a DAE, as well as two methods for performing it, namely the Pantelides method and Pryce's Σ-method. Finally, the steps required for putting the resulting ODE into block triangular form are explained.

Throughout this section, the RLDC2 example introduced in Section 2 is used for illustration purposes, and informal arguments about the parametrization of the whole process are presented, setting the stage for the methods for mDAE analysis described in Section 4.

3.1 Differential-algebraic equations

The general form of a DAE is given by:

$$F(t, x, x', x'', \dots) \tag{2}$$

where F is a system of n_e equations $\{f_1, \dots, f_{n_e}\}$ and x is a finite list of n_v real-valued, smooth enough, variables $\{x_1, \dots, x_{n_v}\}$, functions of the independent variable t. Let x'_j denote the first-order time derivative of x_j, $j = 1, \dots, n_v$. High-order derivatives are recursively defined as usual, and $x_j^{(k)}$ denotes the k-th derivative of x_j. Each f_i depends on some of the variables x_j as well as a finite number of their derivatives.

Let $\sigma_{i,j}$ denote the highest differentiation order of variable x_j effectively appearing in equation f_i, or $-\infty$ if x_j does not appear in f_i. The *leading variables* of F are the variables in the set

$$\left\{ x_j^{(\sigma_j)} \mid \sigma_j = \max_i \sigma_{i,j} \right\} .$$

The *state variables* of F are the variables in the set

$$\left\{ x_j^{(\nu_j)} \mid 0 \leq \nu_j < \max_i \sigma_{i,j} \right\} .$$

A leading variable $x_j^{(\sigma_j)}$ is said to be *algebraic* if $\sigma_j = 0$ (in which case, neither x_j nor any of its derivatives are state variables). In the sequel, v and u denote the leading and state variables of F, respectively.

3.2 Structural Analysis

DAE are a generalization of *ordinary differential equations* (*ODE*), in the sense that it may not be immediate to rewrite a DAE as an explicit ODE of the form $v = G(u)$. The reason is that this transformation relies on the Implicit Function Theorem, requiring that the Jacobian matrix $\frac{\partial F}{\partial v}$ have full rank. This is, in general, not the case for a DAE. Simple examples, like the two-dimensional fixed-length pendulum in Cartesian coordinates [18], exhibit this behaviour.

For a square DAE of dimension n (i.e., we now assume $n_e = n_v = n$) to be solved in the neighborhood of some (v^*, u^*), one needs to find a set of non-negative integers $C = \{c_1, \dots, c_n\}$ such that system

$$F^{(C)} = \{f_1^{(c_1)}, \dots, f_n^{(c_n)}\}$$

can locally be made explicit, i.e., the Jacobian matrix of $F^{(C)}$ with respect to its leading variables, evaluated at (v^*, u^*), is nonsingular. The smallest possible value of $\max_i c_i$ for a set C that

satisfies this property is the *differentiation index* [8] of F, that is, the minimal number of time differentiations of all or part of the equations f_i required to get an ODE.

In practice, the problem of automatically finding a 'minimal' solution C to this problem quickly becomes intractable. Moreover, the differentiation index may depend on the value of (v^*, u^*). This is why, in lieu of numerical nonsingularity, one is interested in the *structural nonsingularity* of the Jacobian matrix, i.e., its almost certain nonsingularity when its nonzero entries vary over some neighborhood. In this framework, the *structural analysis* (*SA*) of a DAE returns, when successful, values of the c_i that are independent from a given value of (v^*, u^*).

A renowned method for the SA of DAE is the *Pantelides method*; however, Pryce's Σ*-method* will also be introduced in what follows, as it is a crucial tool for our works.

3.2.1 Pantelides method

In 1988, Pantelides proposed what is probably the most well-known SA method for DAE [18]. The leading idea of his work is that the structural representation of a DAE can be condensed into a bipartite graph whose left nodes (resp. right nodes) represent the equations (resp. the variables), and in which an edge exists if and only if the variable occurs in the equation.

By detecting specific subsets of the nodes, called *Minimally Structurally Singular* (*MSS*) subsets, the Pantelides method iteratively differentiates part of the equations until a perfect matching between the equations and the leading variables is found. One can easily prove that this is a necessary and sufficient condition for the structural nonsingularity of the system.[7]

The main reason why the Pantelides method is not used in this work is that it cannot efficiently be adapted to general mDAE. As a matter of fact, the adjacency graph of an mDAE has both its nodes and edges parametrized by the subset of modes in which they are active; this, in turn, requires that a parametrized Pantelides method must branch every time no mode-independent MSS is found, ultimately resulting, in the worst case, in the enumeration of modes.

3.2.2 Pryce's Σ-method

Albeit less renowned that the Pantelides method, Pryce's Σ-method [20] is an efficient SA method for DAE, whose equivalence to the Pantelides method has been proved by the author. This method consists in solving two successive problems, denoted by *primal* and *dual*, relying on the Σ*-matrix*, or *signature matrix*, of the DAE F.

This matrix is given by:

$$\Sigma = (\sigma_{i,j})_{1 \leq i,j \leq n} \tag{3}$$

where $\sigma_{i,j}$ is defined as in Section 3.1; as a reminder, $\sigma_{i,j}$ is equal to the greatest integer k such that $x_j^{(k)}$ appears in f_i, or $-\infty$ if variable x_j does not appear in f_i. Note that Σ is the adjacency matrix of a weighted bipartite graph, with structure similar to the graph considered in the Pantelides method, but whose edges are weighted by the highest differentiation orders. The $-\infty$ entries denote non-existent edges.

The *primal problem* consists in finding a *maximum-weight perfect matching* (*MWPM*) in the weighted adjacency graph. This is actually an assignment problem, for the solving of which several standard algorithms exist, such as the push-relabel algorithm [15] or the Edmonds-Karp algorithm [11]. However, none of these algorithms are easily parametrizable, even for applications to mDAE with a fixed number of variables, which strongly influenced both the encoding of the multimode SA problem introduced in Section 4.3 and the solving method described in Section 4.4.

[7] This is done by using the minor expansion formula for computing the determinant of the Jacobian matrix.

The *dual problem* consists in determining $(C, D) = (\{c_1, \ldots, c_n\}, \{d_1, \ldots, d_n\})$, the componentwise minimal solution to a given linear programming problem, defined as the dual of the aforementioned assignment problem. This is performed by means of a *fixpoint iteration* (*FPI*) that makes use of the MWPM found as a solution to the primal problem, described by the set of tuples $\{(i, j_i)\}_{i \in \{1,\ldots,n\}}$:

1. Initialize $\{c_1, \ldots, c_n\}$ to the zero vector.
2. For every $j \in \{1, \ldots, n\}$, $d_j \leftarrow \max_i(\sigma_{i,j} + c_i)$.
3. For every $i \in \{1, \ldots, n\}$, $c_i \leftarrow d_{j_i} - \sigma_{i,j_i}$.
4. Repeat Steps 2 and 3 until convergence is reached.

From the results proved by Pryce in [20], it is known that the above algorithm terminates if and only if it is provided a MWPM, and that the values it returns are independent of the choice of a MWPM whenever there exist several such matchings. In particular, a direct corollary is that the Σ-method succeeds as long as a perfect matching can be found between equations and variables.[8] Figure 6 shows the results given by the Σ-method on the example with both diodes passing.

	i_1	w_2	v_2	w_2	j_1	j_2	v_1	i_2	x_1	x_2	s_1	s_2	u_1	u_2	
K_1	**0**				0	0		**0**							(0)
K_2		**0**					0		**0**				0		(0)
K_3			**0**				**0**						**0**	**0**	(1)
K_4			0	**0**						**0**				0	(0)
L_1		**0**			**1**										(0)
L_2				**0**		**1**									(0)
C_1	**0**						**1**								(0)
C_2			**1**					**0**							(0)
R_1					0				**0**						(0)
R_2						0				**0**					(0)
S_1	**0**										**0**				(0)
S_2								**0**				**0**			(0)
Z_1													**0**		(1)
Z_2														**0**	(1)
	(0)	(0)	(1)	(0)	(1)	(1)	(1)	(0)	(0)	(0)	(0)	(0)	(1)	(1)	

Figure 6: The Σ-matrix (where $-\infty$ entries are omitted) and results of the structural analysis on the RLDC2 example with both diodes passing (i.e., $g_1 = g_2 = \mathbf{T}$). The labels on the left (resp. top) part are the names of the equations (resp. variables). The boxed values represent the chosen solution of the primal problem, i.e., in this case, one of the two existing MWPM. The numbers on the right (resp. bottom) part are the values of the differentiation indices c_i (resp. d_j), i.e., the solution of the dual problem. Entries in bold denote the saturated edges, see Section 3.3.1.

[8] If a perfect matching exists, then a finite number of feasible solutions to the primal problem exist, all of which have finite integer-valued weights. As a result, there is at least one optimal solution to the primal problem, i.e., a MWPM can be found and Pryce's structural analysis succeeds.

Another important result is that, if the Pantelides method succeeds for a given DAE F, then the Σ-method also succeeds for F and the values it returns for C are exactly the differentiation indices for the equations that are returned by the Pantelides method. As for the values of the d_j, being given by $d_j = \max_i(\sigma_{i,j} + c_i)$, they are the differentiation indices of the leading variables in $F^{(C)}$.

Working with this method is natural for our works, since the algorithm for solving the dual problem is easily parametrizable for dealing with multimode systems, as shown in Section 4.5.

3.3 Block triangular decomposition

Once structural analysis has been performed, system $F^{(C)}$ can be regarded, for the needs of numerical solving, as an algebraic system with unknowns $x_j^{(d_j)}$, $j = 1 \dots n$. As such, (inter)dependencies between its equations must be taken into account in order to put it into *block triangular form* (BTF). Three steps are required:

1. the *dependency graph* of system $F^{(C)}$ is generated, by taking into account the perfect matching between equations $f_i^{(c_i)}$ and unknowns $x_j^{(d_j)}$;
2. the *Strongly Connected Components* (*SCC*) in this graph are determined: these will be the *equation blocks* that have to be solved;
3. the *block dependency graph* is constructed as the condensation of the dependency graph, from the knowledge of the SCC; a BTF of system $F^{(C)}$ can be made explicit from this graph.

3.3.1 Dependency graph of $F^{(C)}$

A bipartite graph can be created for $F^{(C)}$, whose left part contains the nodes corresponding to the equations $f_i^{(c_i)}$, and whose right part contains the nodes corresponding to the leading variables $x_j^{(d_j)}$. This graph can be obtained from the adjacency graph of F by only keeping its saturated edges, i.e., any edge between an equation f_i and a variable x_j such that $\sigma_{i,j} + c_i = d_j$, then renaming edges f_i to $f_i^{(c_i)}$ and x_j to $x_j^{(d_j)}$.[9]

This graph is turned into a *directed bipartite graph* (*DBG*) thanks to the perfect matching between equations and variables that was, either computed as a solution to the primal problem in the Σ-method, or obtained by the Pantelides method. Edges that are part of the matching are directed from the equation node to the unknown node; the remaining edges are directed the other way around.

The *dependency graph* is generated by keeping the equation nodes only, and by creating an edge from $f_k^{(c_k)}$ to $f_{k'}^{(c_{k'})}$ if and only if there exists a path from $f_k^{(c_k)}$ to $f_{k'}^{(c_{k'})}$ of length 2, i.e., only passing through one variable node, in the DBG. This (not necessarily connected) graph gives a partial preorder between equations: in broad terms, a path exists from $f_k^{(c_k)}$ to $f_{k'}^{(c_{k'})}$ if and only if the former has to be solved 'before' the latter, since $f_k^{(c_k)}$ is solved for a leading variable whose value is required to solve $f_{k'}^{(c_{k'})}$.

Figure 7 shows the dependency graph for the circuit example with both diodes passing. The creation of a dependency graph for an mDAE is described in Section 4.6.1.

[9] When structural analysis is performed using the Pantelides method, it is also a subgraph of the final graph generated by the method, obtained by only keeping the nodes labeled by the highest differentiations of both the equations and the unknowns.

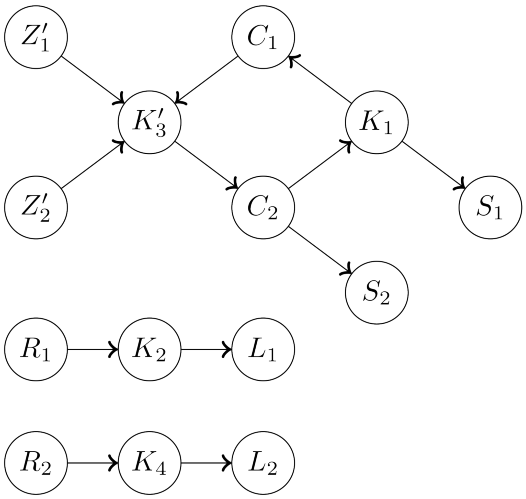


Figure 7: Dependency graph for the circuit example (both diodes are passing). The cycle between nodes K_1, C_1, K'_3 and C_2 indicates that these nodes form an SCC, see Section 3.3.2.

3.3.2 Strongly Connected Components

As there can be mutual dependencies between two or more equations in the dependency graph, such equations need to be solved 'at the same time'; they have to be grouped into *equation blocks*, which are exactly the *SCC* of size larger than 1 in the dependency graph. Other equations correspond to the SCC of size 1.

Tarjan's algorithm [22] is probably the most renowned algorithm for determining the SCC in a directed graph. It is based on a single depth-first search (DFS) in the input graph, which makes it easy to implement and gives it the lowest possible time complexity. Unfortunately, such an approach can require the enumeration of modes for a multimode system, as subcases have to be considered when a mode-dependent edge is traversed. This is why a very different procedure for building equation blocks for an mDAE is introduced in Section 4.6.2.

3.3.3 Block Triangular Form

From the knowledge of the SCC, the dependency graph can be condensed into a directed acyclic graph, called the *block dependency graph*. The edges in this graph define a partial order on the equation blocks; any total ordering consistent with this partial order yields a reordering of equations and variables that puts system $F^{(C)}$ in BTF, which is required for the numerical solving.

Figure 8 shows the block dependency graph obtained from the dependency graph of Figure 7. The adjacency matrix of the system in BTF, obtained from a total ordering on equations consistent with block dependencies, is then given by Figure 9.

The block dependency graph for the DAE of the same circuit, but in another mode (where both diodes are open), is shown in Figure 10. Note that the structure of the DAE system in this mode is very different from the one shown in Figure 8.

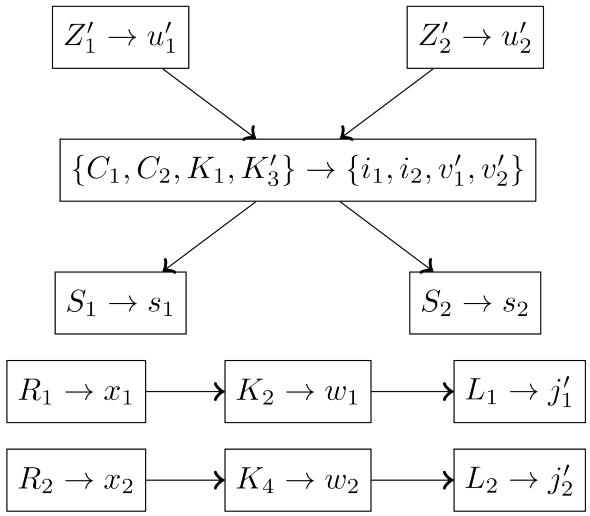


Figure 8: Block dependency graph for the RLDC2 example with both diodes passing (i.e., $g_1 = g_2 = \mathbf{T}$).

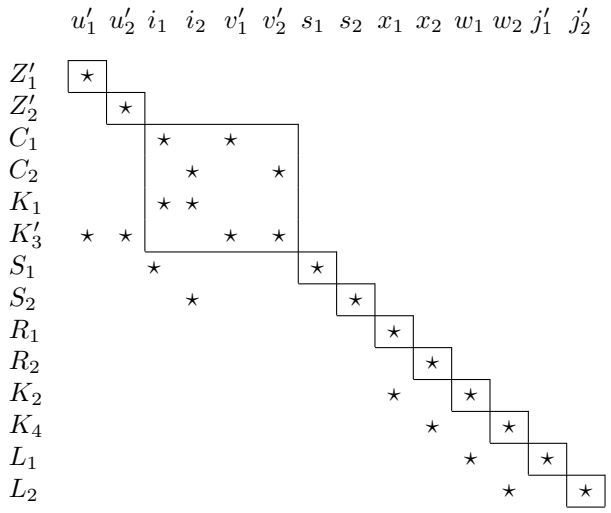


Figure 9: A BTF of the system of equations of the RLDC2 example with both diodes passing. Symbol $\star$ indicates that a variable appears in an equation.

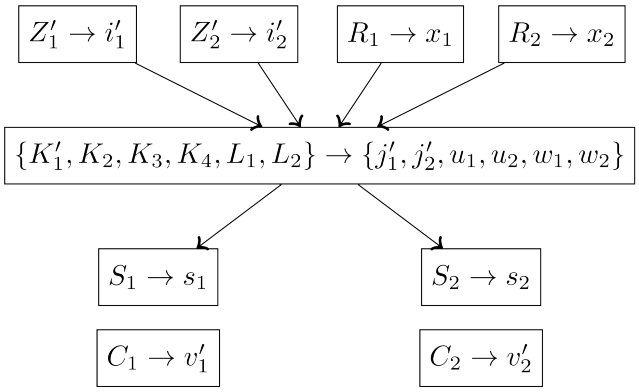


Figure 10: Block dependency graph for the RLDC2 example with both diodes open (i.e., $g_1 = g_2 = \mathbf{F}$).

4 Structural Analysis and Block Dependency of mDAE

This section explains our approach for the solving of both the SA and block dependency problems for mDAE. It relies on *Binary Decision Diagrams* (*BDD*) to efficiently represent, and compute on, functions of Boolean variables [16, 1], more specifically the *Reduced-Ordered* variant (*ROBDD*) by Bryant [6].

We first define the considered class of mDAE, and present the equation language that we use for modeling such systems. As ROBDD are designed for handling Boolean functions of Boolean variables, we then focus on the representation of an mDAE by such functions. Standard operations on those, such as conjunctions and disjunctions, negations, quantifier eliminations, implications, etc. are used to solve both the SA and block dependency problems, which is detailed in the rest of this section, in the same order as in Sections 3.2 and 3.3.

4.1 Multimode DAE

An mDAE differs from a DAE, as defined in Section 3.1, in that the equations f_i become *guarded equations* of the form:

$$\texttt{if } \gamma_i(t, x, x', x'', \dots) \texttt{ then } f_i(t, x, x', x'', \dots) \tag{4}$$

where guard γ_i is a propositional formula. Depending on the truth value of its guard, an equation is *active* or *disabled*. *Guarded variables* can be defined in a similar way, especially for the modeling of reconfigurable systems. If all guards are equal to the constant $\mathbf{T}$, then one recovers a DAE; in other cases, each possible valuation of the set of guards is a mode of the mDAE.

4.2 Representing an mDAE

Although the Modelica language makes it possible to define a subclass of mDAE, it does not enable the user to create models in which variables can be disabled in certain modes.[10] However, such models seem natural for a vast class of reconfigurable systems.

This is why an *ad hoc* input language for our tool, called MEL, was specifically designed to address this shortcoming. This language is similar, in appearance, to a subset of the Modelica language, but addresses some of its shortcomings; in particular, it allows the user to define both guarded equations and guarded variables, whose guards as defined as conditions on variables.

MEL retains only a few important features of Modelica, and introduces new constructs not allowed in the Modelica standard:[11] it allows equations with `if ... then ... else` constructs, but also makes it possible to define varying dimension systems, by enclosing variable declarations and equations in conditional statements. A subset of the grammar of the MEL language is given below, as a guide to the models detailed later (in Figures 13 and 18 of Section 5):

[10] A possible way around this limitation is to force variables that should be disabled in the current mode to zero; however, this can yield wrong results for structural analysis, because of the artificial incidence of these variables in the equations, or even hinder the simulation, as these variables may still be generically used as pivots during the symbolic pivoting occurring at compilation time.

[11] `https://www.modelica.org/documents/ModelicaSpec34.pdf`

mel	::=	| *stmt* ; *mel*
stmt	::=	*decl* | *equ* | *inv*
	|	*cond* | *loop*
decl	::=	*id* : *type* [= *expr*]
equ	::=	*id* : **equation** *expr* = *expr*
inv	::=	**invariant** *expr*
cond	::=	**if** *expr* **then** *mel*
		[**else** *mel*] **end**
loop	::=	**foreach** *id* **in** *enum*
		do *mel* **done**
type	::=	**real** | **boolean** | **integer**
expr	::=	*const* | *id*
	|	*id* (*expr*)
	|	[*expr*] *op* *expr*
	|	(*expr*)
	|	**if** *expr* **then** *expr*
		else *expr*
enum	::=	*expr* .. *expr*

Expressions appearing in equations must be real expressions (Boolean and integer equations are not allowed). Varying dimension systems can be defined by putting variable declarations in a conditional statement. Multimode equations can be defined either by putting them in a conditional statement, or by using a conditional expression inside the equations. Expressions appearing in `invariant` statements must be Boolean propositional formulas: these formulas are used to restrict the set of modes considered during the structural analysis to those that satisfy these statements. External functions are assumed to be differentiable nonlinear functions; their exact definition is not required for structural analysis. Expressions used to define the range of loop statements must be integer expressions.

4.3 Encoding an mDAE with Boolean functions

On the one hand, due to their description in the MEL language, modes, whose set is denoted by M, are naturally represented as valuations of the set of Boolean variables declared via the `boolean` keyword. On the other hand, each equation, variable, and *edge* between an equation and a variable,[12] is associated with its own Boolean variable.

The rationale behind this choice is that this encoding makes it possible to represent subsets of the sets of equations, variables and edges, that are denoted by I, J and E respectively. This comes in handy for edges in particular, as matchings (subsets of E) have to be considered. In the sequel, $\mathcal{P}(X)$ denotes the set of all possible subsets of X. In particular, $\mathcal{P}(E)$ is the set of all possible subsets of edges; due to the chosen encoding, every possible such subset is represented by a valuation of the set of Boolean edge variables. The same holds for equations and variables.

Moreover, a binary encoding of sets E, I and J, although less expensive in terms of memory consumption, could hinder some locality properties that make ROBDD representations of functions more concise. This, of course, assumes that the chosen variable ordering 'preserves locality', in the sense that variables that may interact with one another are close to each other in this ordering; special attention was paid to this specific aspect throughout the design process of our tool.

[12] This terminology was chosen as each edge corresponds to an actual edge in the adjacency graph of the system.

Name	Type	Meaning
χ_M	$M \to \mathbb{B}$	Set of possible modes
χ_I	$M \times I \to \mathbb{B}$	Mode dependency of equations
χ_J	$M \times J \to \mathbb{B}$	Mode dependency of variables
χ_E	$M \times E \to \mathbb{B}$	Mode dependency of edges
σ	$M \times E \to \mathbb{N}$	Values of the $\sigma_{m,i,j}$'s

Table 1: Functions generated from parsing the model.

Table 1 describes the functions that are generated from parsing the original model. Note that a little-endian variable-length binary encoding is used for functions with target set $\mathbb{N}$. The mode dependencies described by χ_I and χ_J are explicit in the model, as well as constraint χ_M on the possible valuations of Boolean mode variables, described thanks to the `invariant` keyword. We say that a valuation $m \in M$ is a *valid mode* if $\chi_M(m)$, an *invalid mode* otherwise.[13] As for χ_E and σ, they are automatically inferred from the model.

Functions $\mathcal{I} : E \to I$ and $\mathcal{J} : E \to J$ respectively return the equation and variable associated to a given edge. Note that consistency conditions for the edges and the equations and variables they involve are automatically checked, namely:

$$\forall e,\ \chi_E(\cdot, e) \Rightarrow (\chi_I(\cdot, \mathcal{I}(e)) \wedge \chi_J(\cdot, \mathcal{J}(e))) \quad .$$

Functions $\mathcal{I}^{-1} : I \to \mathcal{P}(E)$ and $\mathcal{J}^{-1} : J \to \mathcal{P}(E)$ respectively return the set of edges adjacent to a given equation and variable.

4.4 Solving the primal problem

Set of perfect matchings As already mentioned, a matching between equations and variables is encoded as a valuation of Boolean edge variables. As such, a function

$$X : M \times \mathcal{P}(E) \to \mathbb{B}$$

can be defined that describes all perfect matchings in all modes. The uniqueness constraints for equations and variables are translated into functions $\mu\,, \nu : M \times \mathcal{P}(E) \to \mathbb{B}$, defined by:

$$\begin{aligned} \mu &:= \bigwedge_{i \in I} (\chi_I(\cdot, i) \Rightarrow \exists! e \in \mathcal{I}^{-1}(i), e) \ ; \\ \nu &:= \bigwedge_{j \in J} (\chi_J(\cdot, j) \Rightarrow \exists! e \in \mathcal{J}^{-1}(j), e) \ . \end{aligned} \tag{5}$$

These constraints actually do not take into account the fact that edges must be active in a given mode in order to be part of a matching in this mode. This condition is implemented thanks to a function $\Upsilon : M \times \mathcal{P}(E) \to \mathbb{B}$ defined by:

$$\Upsilon := \bigwedge_{e \in E} (e \Rightarrow \chi_E(\cdot, e)) \ . \tag{6}$$

Hence, function X is the conjunction of these three functions:

$$X := \mu \wedge \nu \wedge \Upsilon \ . \tag{7}$$

[13] These notions are structural properties, independent from the dynamical property of reachability of a mode: a mode may be valid yet unreachable.

Weight function For pruning out from X every matching whose weight is not maximal, we define a function

$$\omega : M \times \mathcal{P}(E) \to \mathbb{N}$$

yielding the weight of any subset of edges in any mode. This is performed thanks to parametrized arithmetic operations on variable-length binary integers, combined with a parametrized `if-then-else` operator. More precisely, the following operation is iterated on all edges $e \in E$, starting from the zero function:

$$\omega \leftarrow \texttt{if } e \texttt{ then } \omega + \sigma(\cdot, \mathcal{I}(e), \mathcal{J}(e)) \texttt{ else } \omega \ .$$

Set of MWPM and choice of one MWPM per mode Instead of determining only one solution of the primal problem for each mode, it is actually easier to first compute, from the function X describing all perfect matchings, the function

$$S : M \times \mathcal{P}(E) \to \mathbb{B}$$

describing all MWPM. Determining S is essentially performing an $arg\,max$ of the weight function ω in order to pick specific elements of $M \times \mathcal{P}(E)$ among the ones that satisfy X. This operation is performed via a specific algorithm, that we describe hereafter as it is a subtle and crucial part of the tool.

Let $(\omega_k : M \times \mathcal{P}(E) \to \mathbb{B})_{k=0\ldots N-1}$ be the (parametrized) digits of function ω. The value of N is implicitly known, as it is the length of the array of ROBDD representing ω. Let $S_N \equiv X$; functions $\omega_k^{\max} : M \to \mathbb{B}$ and $S_k : M \times \mathcal{P}(E) \to \mathbb{B}$ are defined, for k ranging from $N-1$ down to 0, by:

$$\begin{aligned} \omega_k^{\max} &:= \exists E.(S_{k+1} \wedge \omega_k) \ ; \\ S_k &:= S_{k+1} \wedge (\omega_k \Leftrightarrow \omega_k^{\max}) \ . \end{aligned}$$

Essentially, $\omega_k^{\max}$ describes the modes in which the bit at position k in the weight of a matching in S_{k+1} can be equal to 1. In these modes only, matchings whose weights have their k-th bit equal to 0 have to be pruned out, hence the equivalence $\omega_k \Leftrightarrow \omega_k^{\max}$.

As a result, the sequence S_N, S_{N-1}, …, S_0 represents a decreasing nested sequence of sets of matchings, obtained by iteratively pruning out those matchings whose weight cannot be maximal. The function describing the set of MWPM is then given by $S \equiv S_0$. From this function, it is possible to derive a function

$$T : M \times \mathcal{P}(E) \to \mathbb{B}$$

describing a choice of one MPWM per mode without having to enumerate the modes. The computation of function T from function S is efficiently performed thanks to an inductive algorithm on BDD, that itself relies on memoization techniques.

For conveniency, a function $T_e : M \to \mathbb{B}$ is generated for every edge $e \in E$, indicating the modes in which edge e is part of the chosen MWPM. It is simply defined by:

$$T_e := \exists E, \ (T \wedge e) \ .$$

Once again, a specialized inductive algorithm was designed for optimizing this computation.

Singularity diagnostics The structural singularity of the system in one or several modes can be easily diagnosed from the knowledge of function X, describing all perfect matchings. This is performed via quantifier elimination; let $\mathcal{N} : M \to \mathbb{B}$ be defined by:

$$\mathcal{N} := (\chi_M \Rightarrow (\exists E, \ X)) \ . \tag{8}$$

Name	Type	Meaning
X	$M \times \mathcal{P}(E) \to \mathbb{B}$	Set of perfect matchings
ω	$M \times \mathcal{P}(E) \to \mathbb{N}$	Weight function
S	$M \times \mathcal{P}(E) \to \mathbb{B}$	Set of MWPM
T	$M \times \mathcal{P}(E) \to \mathbb{B}$	Choice of one MWPM per mode
T_e	$M \to \mathbb{B}$	Edge e is in the chosen MWPM
$\mathcal{N}$	$M \to \mathbb{B}$	Modes in which the system is SNS

Table 2: Functions computed during the solving of the primal problem.

Function $\mathcal{N}$ describes the union of the modes in which the system is *structurally nonsingular* (*SNS*) and the invalid modes. Whenever $\mathcal{N}$ is equal to the constant $\mathbf{T}$, it is known that structural analysis will succeed in all valid modes, following the remarks about the Σ-method in Section 3.2.2.

Otherwise, any clause satisfying $\neg\mathcal{N}$ describes a valid mode in which the system is structurally singular. By using partial evaluations for this valuation of the Boolean mode variables, and adapting the algorithms described above, one is able to retrieve information about maximal matchings in this mode, including which variables (resp. equations) cannot be matched. Information can be returned to the user, providing him/her with hints for redesigning the model.

Table 2 summarizes the functions computed for solving the primal problem and checking the nonsingularity of the model.

4.5 Solving the dual problem

Steps 2 and 3 from the FPI algorithm presented in Section 3.2.2 have to be adapted so that they compute functions $\mathbf{c}_i : M \to \mathbb{N}$ (for every $i \in I$) and $\mathbf{d}_j : M \to \mathbb{N}$ (for every $j \in J$). For both simplicity and conciseness of the ROBDD representation, a c_i (resp. d_j) is set to 0 in those modes in which equation f_i (resp. variable x_j) is disabled. As such, the parametrized FPI also has to check that the conditions enforced by functions χ_E, χ_I and χ_J are satisfied.

Using a parametrized max function, as well as arithmetic operations and the `if-then-else` operator introduced in Section 4.4, the parametrized FPI reads as follows:

1. Initialize $\mathbf{c}_1, \ldots, \mathbf{c}_n$ to the zero function.

2. For every $j \in \{1, \ldots, n\}$,

$$\mathbf{d}_j \leftarrow \texttt{if } \chi_J(j) \texttt{ then } \max_{e \in \mathcal{J}^{-1}(j)} \left\{\texttt{if } \chi_E(e) \texttt{ then } \mathbf{c}_{\mathcal{I}(e)} + \sigma(\cdot, e) \texttt{ else } 0\right\} \texttt{ else } 0\ .$$

3. For every $i \in \{1, \ldots, n\}$,

$$\mathbf{c}_i \leftarrow \texttt{if } \chi_I(i) \texttt{ then } \max_{e \in \mathcal{I}^{-1}(i)} \left\{\texttt{if } \chi_J(\mathcal{J}(e)) \wedge T(e) \texttt{ then } \mathbf{d}_{\mathcal{J}(e)} - \sigma(\cdot, e) \texttt{ else } \mathbf{c}_i\right\} \texttt{ else } 0\ .$$

4. Repeat Steps 2 and 3 until convergence is reached.

4.6 Multi-mode block dependency

This section follows the main three steps for building the block dependency graph, as described in Section 3.3.

4.6.1 Parametrized dependency graph

An adjacency graph is generated whose nodes are named according to the original equations f_i and variables x_j, without distinguishing between their successive time derivatives. An edge e between nodes f_i and x_j only exists in this graph for those modes in which e is both active and saturated (see Section 3.3.1).

For every $e \in E$, let $\Xi_e : M \to \mathbb{B}$ be the function representing the set of modes in which edge e is in the adjacency graph. This function is given by:

$$\Xi_e := \chi_E(\cdot, e) \ \wedge \ \left(\mathbf{d}_{\mathcal{J}(e)} = \mathbf{c}_{\mathcal{I}(e)} + \sigma(\cdot, e)\right) \ .$$

As the graph is then directed according to the chosen MWPM, function T is used in order to generate a parametrized preorder relation $\rightsquigarrow$: $I \times I \to (M \to \mathbb{B})$ essentially equivalent to the dependency graph described in Section 3.3.1. For every pair of edges e, e' such that $\mathcal{J}(e) = \mathcal{J}(e')$, a path of length 2 exists between $\mathcal{I}(e)$ and $\mathcal{I}(e')$ (i.e., the variable matched with equation e appears in e') if:

$$(\Xi_e \wedge T_e) \wedge (\Xi_{e'} \wedge \neg T_{e'}) \ .$$

If the result of this computation is not the *false* constant $\mathbf{F}$, then function $(\mathcal{I}(e) \rightsquigarrow \mathcal{I}(e'))$ is replaced with its disjunction with this result. This process is iterated on all ordered pairs of edges in order to get relation $\rightsquigarrow$.

4.6.2 Equation blocks

As mentioned in Section 3.3.2, Tarjan's algorithm [22] is, in the single-mode DAE setting, an efficient solution for condensing the dependency graph and getting a total ordering on the resulting equation blocks; however, performing a DFS on the dependency graph of an mDAE can require considering subcases when new edges are taken into account, resulting in the worst case in the enumeration of all modes. For the creation of the mode-dependent equation blocks, one may go back to a more immediate definition of the SCC.

The transitive closure $\rightsquigarrow^*$ of relation $\rightsquigarrow$ can be easily computed in an iterative manner; it exhibits a reachability relation in $\rightsquigarrow$, as $i \rightsquigarrow^* i'$ exactly means that there exists a nonnegative integer K and a sequence $i_0, i_1, \ldots, i_K$ such that $i_0 = i$, $i_{k-1} \rightsquigarrow i_k$ for all $k = 1, \ldots, K$, and $i_K = i'$. (In particular, $i \rightsquigarrow^* i$ for all i.)

Following this, two nodes i and i' belong to the same SCC if and only if there are mutually reachable, i.e., $(i \rightsquigarrow^* i') \wedge (i' \rightsquigarrow^* i)$. This defines an equivalence relation indicating, for every i and i', the set of modes in which the leading equations derived from f_i and $f_{i'}$ belong to the same SCC.

A recursive algorithm is used to build the mode-dependent equation blocks, by taking into account the orders of differentiation of the leading equations given by the c_i.

The prospect of pushing towards the development of a compiler also led us to distinguish equation blocks with respect to the unknowns that are evaluated from them, given by the information of the chosen MWPM (given by function T) and the differentiation orders for variables (given by functions d_j).

Furthermore, an equation block also comes with the list of all variables whose values have to be provided as inputs for solving it. This is basically done by exploring, for every equation $f_i^{(k)}$

appearing in the equation block, every edge $e \in \mathcal{I}^{-1}(i)$ and checking the conditions under which the corresponding variable $x_{\mathcal{J}(e)}$ or its time derivatives can be inputs. Functions $d_{\mathcal{J}(e)}$ and σ are involved in determining the values of l for which $x^{(l)}_{\mathcal{J}(e)}$ is an actual input for the block. Of course, every block comes with a function $M \to \mathbb{B}$ describing those modes in which it has to be evaluated.

In the end, several partial duplications may take place, for distinguishing between blocks that contain the same equations with the same orders of differentiation, but different lists of variables as either inputs or outputs. However, such block duplications appear to be seldom in practice, thus having negligible impact on computational times and memory consumption.

4.6.3 Block dependency graph

Put together, relation $\rightsquigarrow$ and the list of blocks contain all the information of the mode-dependent condensation of the dependency graph; our algorithm outputs all useful data for reconstructing it and/or evaluating it in a given mode.

An illustration is given in Figure 11, giving the block dependency graph for the RLDC2 model. This graph was directly created by using GraphViz on the `.dot` file generated by Isam-DAE; for every valuation of the mode variables, it yields the expected dependency graph in the corresponding mode of the system (in particular, those given in Figures 8 and 10).

The BTF of the system can then be found at runtime for any given mode, for instance by evaluating the block dependency graph in this mode and performing a topological sort on the resulting graph; this is out of the scope of our study. Note, however, that all equation blocks can be turned into efficient simulation code at compile time, so that the computational overhead due to a mode switching at runtime is minimized.

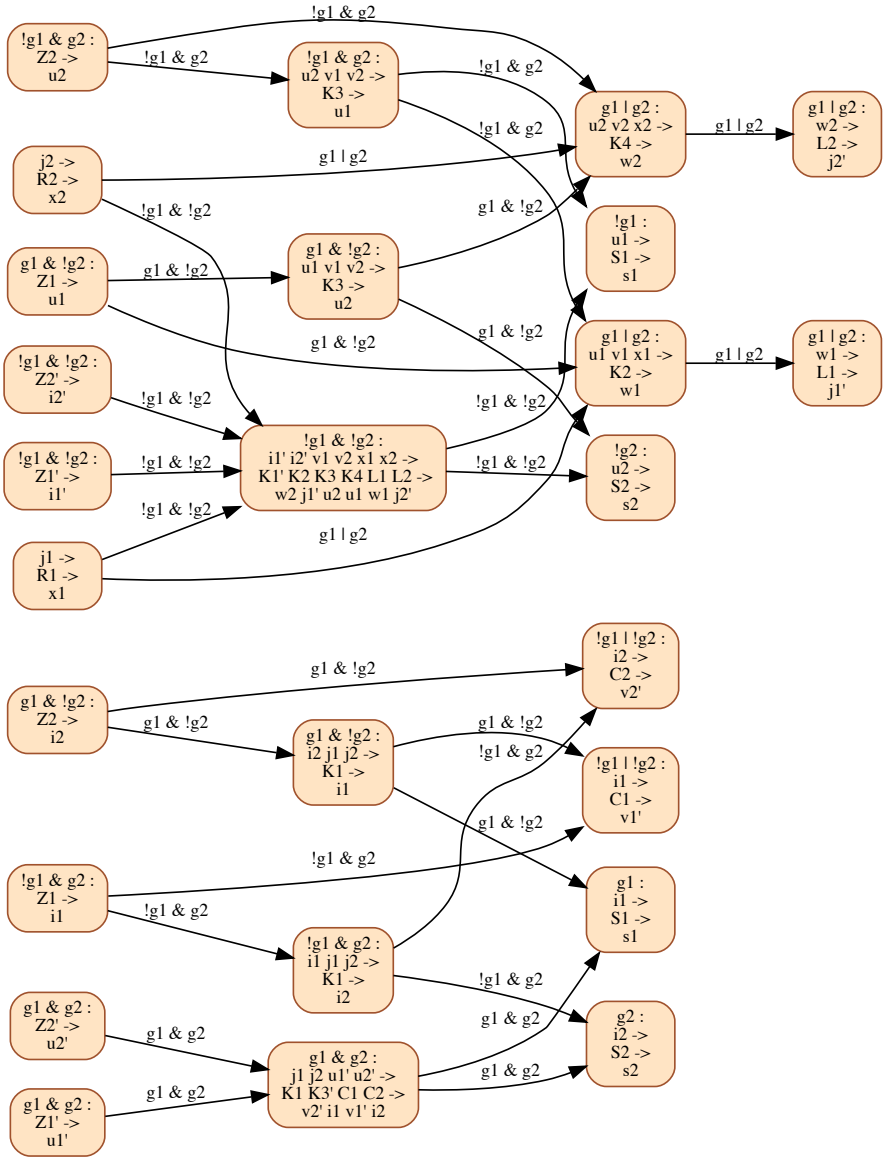


Figure 11: Block dependency graph of the RLDC2 model, generated by IsamDAE. Vertices are labeled $p : R \to B \to W$, where: p is a propositional formula defining in which modes the block is evaluated; R is the set of variables to read; B is the set of equations of the block; W is the set of variables to write. Edges are labeled by a propositional formula, defining in which modes the dependency applies.

5 Experiments

This section details the experimental results obtained with IsamDAE on three different models. In particular, we are interested in the growth of computational times according to the value given to the size parameter of scalable models.

The reason is that all computations on ROBDD have an exponential worst-case complexity. As a result, the computational complexity of the proposed method is bound to be, in the worst case, linear in the number of modes of the mDAE, which is the current state of the art. However, system models are often sparse and show some regularity, which may enable our method to exhibit sub-exponential time and space complexity for models where blocks of equations are localized. This last notion means that, not only the blocks are of bounded size, but also the adjacency graph has a bounded treewidth [21].

The first two models are variants of a simple model building; not only do we exhibit sub-exponential computational times on one of these models, but we also provide explanations for the difference in behavior between these models, thereby hinting at possible modeling principles for multimode systems.

The last model is that of a rail brake system, known as the Westinghouse brake system, on which, once again, sub-exponential computational times are obtained.

CPU times were obtained on a MacBook Pro equipped with an Intel Core i7 processor at 2.9GHz and 16GB of RAM.

5.1 The building model

The model A single-story building is made of N rooms of the same size, numbered from 1 to N, all connected to a single corridor by simple doors. Intake and exhaust vents are present in every room, with constant external pressures. This building is represented by Figure 12.

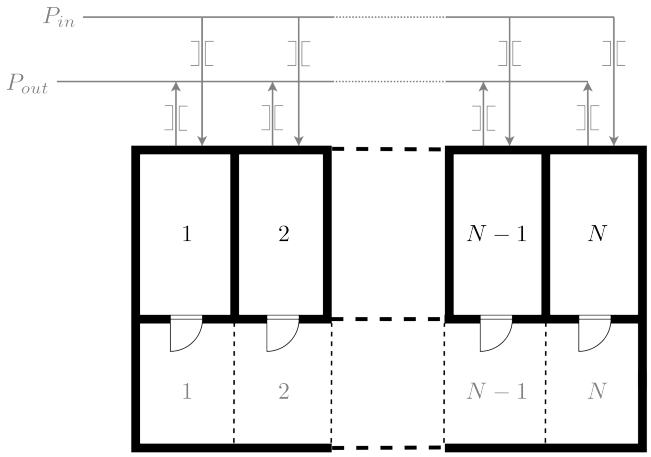


Figure 12: Schematics of a simplified single-story building model.

Under the assumption that the temperature is homogeneous in each room and in each corridor element, the variables taken into account are the pressures, temperatures, enthalpies and air masses in every room and in every corridor element, as well as the mass flows and heat flows through all doors, intake vents and exhaust vents.

For modeling conveniency, the pressure and temperature in 'ghost' corridor element 0 are defined and forced to be equal to those in corridor element 1. For similar reasons, mass and heat flows at the other end of the corridor are defined and set to 0 by two equations. Thanks to these few extra variables, the N sections of the building can be described by the same set of equations, as some of them imply, either the temperature and/or pression in the previous section, or the heat and/or mass flux from or to the next section.

Three mode variables are associated with every $i \in \{1; \ldots; N\}$. The first one states whether the door of room i is open; the second one indicates the direction of the air flow between room i and the corridor; the third one indicates the direction of the air flow between elements $i-1$ and i of the corridor.

Invariants are defined for stating the following conditions, that are natural for the model: there can be no mass or heat flow from section 0; if the door of room i is closed, there can be no mass or heat flow from this room to the corresponding corridor element. Because of these constraints on the modes, there is a grand total of $3^N \times 2^{N-1}$ modes.

The model exists in two variants: one in which air is considered incompressible, and one in which it is compressible. Both variants are meaningful, from a physicist's point of view, simply because pressure differences between rooms are small relatively to atmospheric pressure. Only a few equations differ, namely the equations linking room/corridor temperatures with the mass of air contained in a room or corridor element; pressure only appears in these equations in the compressible case. Both variants have $16N + 5$ equations and variables.

Figure 13 shows the incompressible model, written in the MEL language. All variable declarations are omitted for conciseness. `Pr` (resp. `Pc`) is a constant, equal to the volume of a room (resp. a corridor element). `Pin` (resp. `Pout`) is a constant, equal to the relative pressure of the intake (resp. exhaust) ventilation circuit. `Tin` is a constant, equal to the temperature of the intake air. Finally, `rho`, `enthalpy`, `work`, `flow_vent`, `flow_corridor` and `door` are external functions, whose exact expressions are of no interest in the context of structural analysis. The only assumption on these functions is that they are differentiable.

The compressible variant of this model is easily created by replacing equations `rtm[i]` and `ctm[i]`, describing the total mass of air in each room and each corridor element, so that the density becomes a function, not only of the temperature, but also of the pressure, in this room or element.

Experimental results In the incompressible variant (see Figure 14), the number of equation blocks grows as an exponential of parameter `N` (the number of rooms), and the maximum size of the blocks increases linearly in `N`. This comes from the fact that the corridor pressures `Pc[i]` are algebraic variables, computed by equation blocks that depend on the state of every door (`open[i]`) and the pressure `Pr[i]` in every room with an open door. As a result, the 2^N possible states of the set of doors result in 2^N different equation blocks involving the pressures in the corridor elements, the largest of whose relates variables from all the corridor elements and all the rooms.

On the other hand, Figure 15 shows that the compressible variant of the model behaves in a completely different manner. Block sizes are bounded, and the number of blocks is an affine function of parameter `N`. This comes from the fact that corridor pressures `Pc[i]` are now state variables, and air flows only depend on the pressures of the neighboring corridor elements and rooms. This locality property explains why the structural analysis of the model scales up. As a matter of fact, computational times are roughly proportional to N^3, despite an exponential number of modes.

```
// Global equations

time : equation der(t) = 1.0;

// temperature and pressure at one end of the corridor
plug_Tc : equation Tc[0] = Tc[1];
plug_Pc : equation Pc[0] = Pc[1];

// mass and heat flows at the other end
plug_mu_c_2 : equation mu_c[N+1] = 0.0;
plug_eta_c : equation eta_c[N+1] = 0.0;
invariant !direction[1];

// Rooms and corridor elements

foreach i in 1 .. N do
  // Equations for room i

  // mass balance equation
  rmb[i] : equation der(Mr[i]) = mu_in[i] - mu_out[i] + mu_door[i];
  rimf[i] : equation mu_in[i] = flow_vent(Pin - Pr[i]); // intake flow
  romf[i] : equation mu_out[i] = flow_vent(Pr[i] - Pout); // exhaust flow
  open[i] : boolean = door(t); // boolean defining whether door is open
  if open[i] then
      // if door is open room pressure equals corridor pressure
      dop[i] : equation Pr[i] = Pc[i]
    else
      // if door is closed air flow through door is zero
      dcf[i] : equation mu_door[i] = 0
    end;

  // heat balance
  reb[i] : equation der(Er[i]) = eta_in[i] - eta_out[i] + eta_door[i] + work(t);
  // intake heat flow
  rief[i] : equation eta_in[i] = mu_in[i] * enthalpy(Tin);
  // exhaust heat flow
  roef[i] : equation eta_out[i] = mu_out[i] * enthalpy(Tr[i]);
  // boolean defining the flow direction
  outgoing[i] : boolean = mu_door[i];
  // if door is closed, flow can not be going out of the room
  invariant open[i] | !outgoing[i];
  rdef[i] : equation eta_door[i] = mu_door[i] *
                                   enthalpy(if outgoing[i] then Tr[i] else Tc[i]);
  // air mass as a function of the temperature
  rtm[i] : equation Mr[i] = Vr * rho(Tr[i]);
  rte[i] : equation Er[i] = Mr[i] * enthalpy(Tr[i]);

  // Equations for corridor element i

  // mass balance equation
  cmb[i] : equation der(Mc[i]) = mu_c[i] - mu_c[i + 1] - mu_door[i];
  // heat balance equation
  ceb[i] : equation der(Ec[i]) = eta_c[i] - eta_c[i + 1] + eta_door[i];
  // air mass as a function of the temperature
  ctm[i] : equation Mc[i] = Vc * rho(Tc[i]);
  cte[i] : equation Ec[i] = Mc[i] * enthalpy(Tc[i]);
  cmf[i] : equation mu_c[i] = flow_corridor(Pc[i - 1] - Pc[i]);
  // boolean defining the flow direction in the corridor
  direction[i] : boolean = mu_c[i];
  cef[i] : equation eta_c[i] = mu_c[i] *
                               enthalpy(if direction[i] then Tc[i-1] else Tc[i]);
done;
```

Figure 13: The building model, shortened by omitting variable declarations.

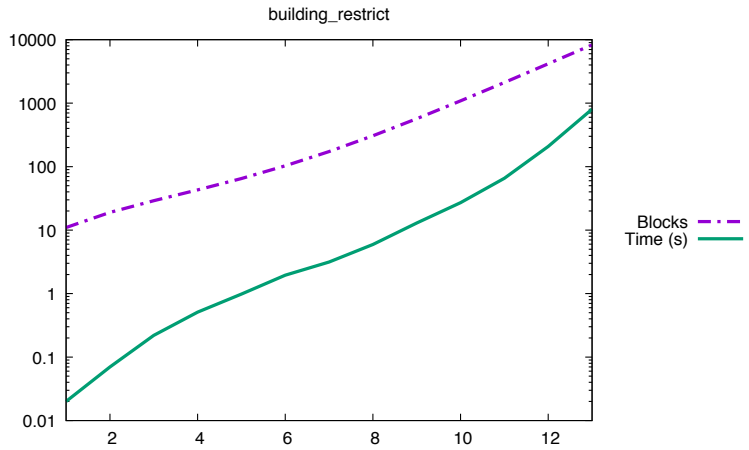


Figure 14: Number of blocks and CPU time for the structural analysis of the building model with incompressible air.

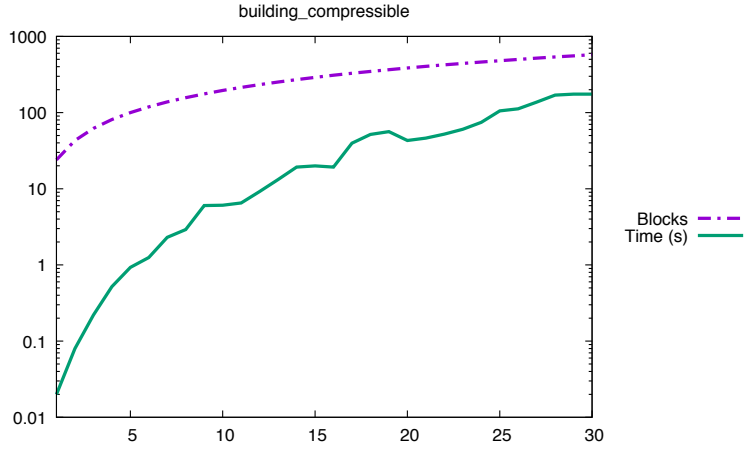


Figure 15: Number of blocks and CPU time for the structural analysis of the building model with compressible air.

5.2 A Westinghouse rail brake system model

The model The Westinghouse brake system is the first fail-safe air brake system for railways; it was patented by George Westinghouse in 1869. With this system, a brake is only released when the pressure in the associated air reservoir reaches a given threshold, ensuring that the brakes engage in case of an accidental pressure loss. A diagram of the subsystem at the level of each single railcar is given in Figure 16.

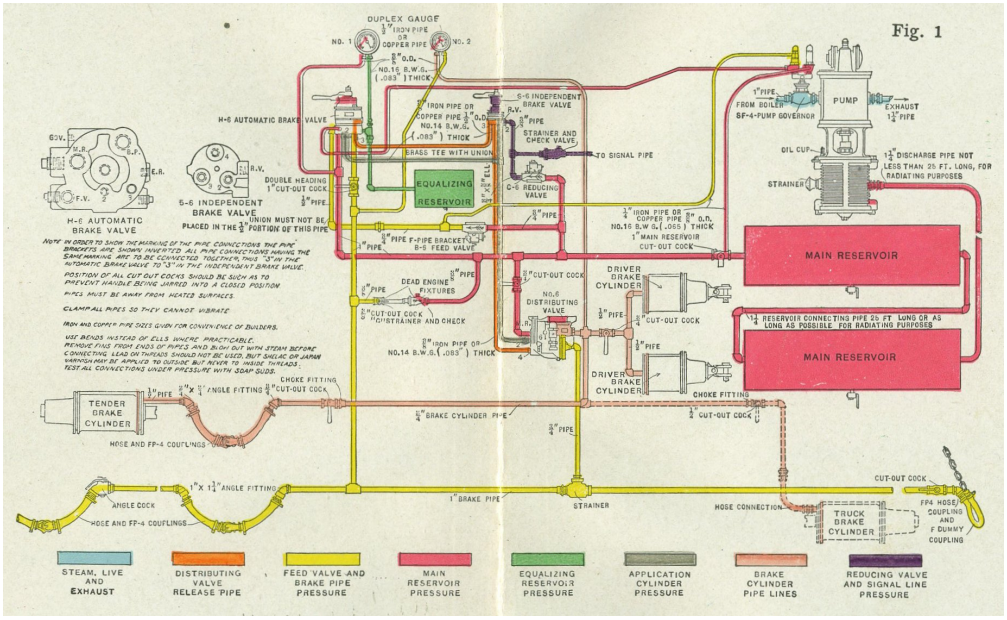


Figure 16: Original schematics of the Westinghouse brake system at the level of the engine and tender.

In the model shown in Figure 18, N railcars are connected. The variables are the pressures `Pb[i]` in the reservoirs, `Pr[i]` in the brake lines, and `Pt[i]` in the connectors of the brake lines; six different mass flows for each railcar, whose names begin with the letter `f`; the brake force `b[i]`, and the position `x[i]` of the piston. It is assumed that air temperature is constant. The pressures and mass flows between railcar N and a 'ghost' railcar $N+1$ are defined so that all railcars can be described with the same set of equations.

`control` is an external function of time representing the control of the brakes, applied at the level of railcar 1. `flow` is an external function relating air flows to pressure drops. `V`, `S`, `L`, `K`, `F1`, `F2`, `Rho`, `P0` are constants.

This model has 2^N modes, and $11N + 2$ equations in each mode.

Experimental results The results obtained with this model are shown in Figure 17. Good performances are obtained mainly because the block sizes are bounded and the number of blocks is an affine function of the number of railcars `N`. It turns out that no block spans over more than three adjacent railcars.

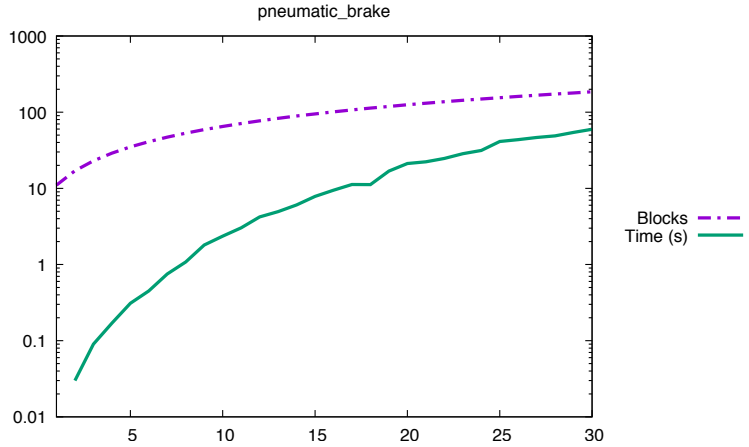


Figure 17: Number of blocks and CPU time for the Westinghouse brake model.

```
// Plug equations at one end of the train

dt : equation der(t) = 1.0; // time passes
plug1 : equation Pb[1] = control(t); // brakes are applied according
                                     // to a function of time

// Railcars
foreach i in 1 .. N do
    // Mass balance equations

    // at the brake line
    me1[i] : equation fb[i] - fb[i + 1] - fv[i] - fcl[i] + fl[i] = 0;
    // at the reservoir
    me2[i] : equation fv[i] - fch[i] - fl[i] - ft[i] = 0;

    // Flow equations

    // input mass flow
    feb1[i] : equation fb[i] - F1*flow(Pb[i]-Pr[i]) = 0;
    // output mass flow
    feb2[i] : equation fb[i + 1] + F1*flow(Pb[i + 1]-Pr[i]) = 0;
    // leakage mass flow
    fel[i] : equation fl[i] - F2*flow(Pt[i]-Pr[i]) = 0;

    // Dynamics of the reservoir

    // pressure equation of the reservoir
    mer[i] : equation ft[i] = Rho*V*der(Pt[i])/P0;

    // Dynamics of the piston

    mepl[i] : equation Rho*S*(der(x[i])*Pr[i] +
                             (x[i]-L)*der(Pr[i])) + P0*fcl[i] = 0;
    meph[i] : equation Rho*S*(der(x[i])*S*Pt[i] +
                             x[i]*der(Pt[i])) - P0*fch[i] = 0;

    // Brake force and mechanical linkage movement

    // brake force equation
    pb[i] : equation S*(Pt[i]-Pr[i]) = b[i];
    // elastic deformation of the mechanical linkage
    px[i] : equation b[i] = K*x[i];

    // One way valve

    open[i] : boolean = last(fv[i]); // two modes: true = valve is open;
                                     //             false = valve is closed

    if open[i] then
      // when open, the valve opposes no pressure drop
      ve1[i] : equation Pr[i] = Pt[i];
    else
      // when closed, no air passes through the valve
      ve2[i] : equation fv[i] = 0;
    end
done;

// Plug equation at the other end of the train

plug2 : equation fb[N + 1] = 0;
```

Figure 18: The Westinghouse railway brake model, shortened by omitting variable declarations.

6 Conclusion

We presented a method for performing the structural analysis and computing the block-triangular decomposition of multimode DAE systems, for all modes at once. This approach is motivated by the need for a static analysis process for mDAE that makes it possible, at compile time, to both reliably check the provided model and compute all the information needed for the efficient generation of simulation code.

An *ad hoc* equation-based language is used for describing variable-structure multimode models. We showed how implicit functional representations are used for encoding an mDAE model, and what computations are performed in order to check the structural nonsingularity of the model, perform its structural analysis all-modes-at-once (by generalizing J. Pryce's Σ-method), and compute a conditional dependency graph yielding all necessary information for efficient simulation code generation.

The described process was implemented in OCaml, and its efficiency was demonstrated on a model of heat and mass flows in a building, for which the number of modes is exponential in the number of rooms. On the compressible variant of this model, both the computational times and the size of the resulting dependency graphs appeared to be polynomial in the size of the model, thus overcoming the challenge of having to deal with an exponential number of modes.

The structural analysis of mode switchings in a similar implicit framework is a topic of ongoing research. Perspectives in the shorter term include the design of a modular approach for the computations, so that subsystems may be solved independently before merging the partial solutions into global solutions; we believe that this is a key step towards the scalability of the software. We are also interested in the solving of graph-theoretical problems in order to optimize variable allocation: the choice of a variable ordering is crucial for ROBDD computations to be efficient, which should not concern the designer of the model itself.

References

[1] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, jun 1978.

[2] A. Benveniste, T. Bourke, B. Caillaud, J.-L. Colaço, C. Pasteur, and M. Pouzet. Building a hybrid systems modeler on synchronous languages principles. *Proceedings of the IEEE*, 106(9):1568–1592, 2018.

[3] A. Benveniste, B. Caillaud, H. Elmqvist, K. Ghorbal, M. Otter, and M. Pouzet. Structural analysis of multi-mode DAE systems. In G. Frehse and S. Mitra, editors, *Proc. of the 20th International Conference on Hybrid Systems: Computation and Control, HSCC 2017, Pittsburgh, USA, April 18-20, 2017*, pages 253–263. ACM, 2017.

[4] A. Benveniste, B. Caillaud, H. Elmqvist, K. Ghorbal, M. Otter, and M. Pouzet. Multi-mode DAE models - challenges, theory and implementation. In B. Steffen and G. J. Woeginger, editors, *Computing and Software Science - State of the Art and Perspectives*, volume 10000 of *Lecture Notes in Computer Science*, pages 283–310. Springer, 2019.

[5] D. Broman and J. G. Siek. Modelyze: a gradually typed host language for embedding equation-based modeling languages. Technical Report UCB/EECS-2012-173, EECS Department, University of California, Berkeley, Jun 2012.

[6] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, aug 1986.

[7] B. Caillaud, M. Malandain, and J. Thibault. Implicit structural analysis of multimode DAE systems. In *23rd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2020)*, Sydney, Australia, April 2020.

[8] S. L. Campbell and C. W. Gear. The index of general nonlinear DAEs. *Numerische Mathematik*, 72(2):173–196, dec 1995.

[9] A. D. Domínguez-García and S. Trenn. Detection of impulsive effects in switched DAEs with applications to power electronics reliability analysis. In *Proceedings of the 49th IEEE Conference on Decision and Control, CDC 2010, December 15-17, 2010, Atlanta, Georgia, USA*, pages 5662–5667. IEEE, 2010.

[10] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 2017.

[11] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, apr 1972.

[12] H. Elmqvist, T. Henningsson, and M. Otter. Systems modeling and programming in a unified environment based on Julia. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications*, pages 198–217, Cham, 2016. Springer.

[13] H. Elmqvist, S. E. Mattsson, and M. Otter. Modelica extensions for multi-mode DAE systems. In *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*. Linköping University Electronic Press, mar 2014.

[14] P. Fritzson. *Principles of Object Oriented Modeling and Simulation with Modelica 3.3*. John Wiley & Sons, Inc., nov 2014.

[15] A. V. Goldberg and R. Tarjan. A new approach to the maximum flow problem. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing - STOC '86*. ACM Press, 1986.

[16] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38(4):985–999, jul 1959.

[17] D. Liberzon and S. Trenn. Switched nonlinear DAE: Solution theory, Lyapunov functions, and stability. *Automatica*, 48(5):954–963, 2012.

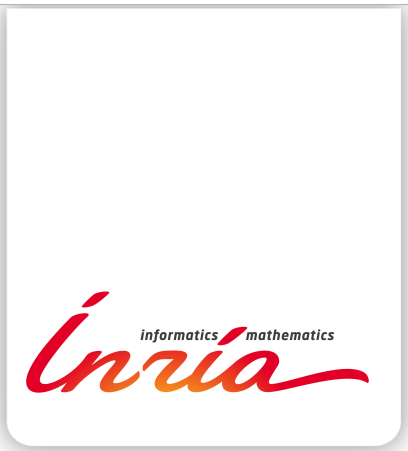
[18] C. C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231, March 1988.

[19] A. Pothen and C.-J. Fan. Computing the block triangular form of a sparse matrix. *ACM Trans. Math. Softw.*, 16(4):303–324, 1990.

[20] J. D. Pryce. A simple structural analysis method for DAEs. *BIT Numerical Mathematics*, 41(2):364–394, March 2001.

[21] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, September 1986.

[22] R. Tarjan. Depth-first search and linear graph algorithms. In *12th Annual Symposium on Switching and Automata Theory*. IEEE, oct 1971.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399